XV6 ASSIGNMENT REPORT

ABHAY SHANKAR K: CS21BTECH11001

1. Demand Paging

1.1. `exec.c.` :

- In `exec.c`, the function `allocuvm` is called to allocate space for the code and data segments.
- Specifically, it allocates `ph.memsz` bytes, where `ph` is the program header and memsz is the amount of memory required for both code and data.
- It is evident, after examining the values of these variables for several user programs, as well its usage in the subsequent `loaduvm()` call that `ph.filesz` is the memory needed for the code segment alone. Thus, we change the argument of allocuvm from `ph.vaddr + ph.memsz` to `ph.vaddr + ph.filesz`.
- This has the side effect of not reserving any virtual memory space for the data segment - the stack is allocated immediately after the code. To rectify this, we add the following: `sz += ph.memsz - ph.filesz`.
- This statement causes the stack memory to be allocated sufficiently many pages after the code segment so as to accomodate the entirety of the data segment within it.

1.2. `trap.c.` :

- We add a case within the trap handler for when the trap number is `T_PGFLT = 14`. This calls the `pgflt_handler()` function.
- We check the value of the present bit in the page table entry corresponding to the faulting page - if it is 0 (i.e. entry does not exist) then we allocate one page of memory corresponding to that virtual address using `single_alloc()`, which is a wrapper around `allocuvm`.
- Within allocuvm, the physical address of the frame is obtained and the page table is updated accordingly.

2. Copy-on-write

2.1. `kalloc.c.` :

- We created a new glocal array `refs`, which holds the reference count for each frame in memory.
- We updated `kalloc` and `kfree` to increment and decrement the reference count of they frame they allocate and free, respectively.
- Two new functions were created to lock and unlock the spinlock within `kmem`. They are used each time the reference count is changed in any file.

2.2. `vm.c.` :

- The function `walkpgdir()` was presenting errors, so a less general function, `walk_wrap()` was implemented.
- This function is used to obtain the physical address of a page from a page table.
- The function `copyuvm` is altered - it no longer allocates new pages, merely maps the physical addresses of the frames allocated to the parent process onto the page table of the child process.
- Before doing so, it resets the write bit for all page table entries in the parent process and increment the reference count for all frames allocated to the parent process.
- When any process attempts to write to the page, it triggers a page fault.

2.3. `trap.c.` :

- Within `pgflt_handler()`, we add another case.
- If the page fault is indeed caused by a copy-on-write event, (which we check for by ensuring that the user bit is set and the write bit is reset), we allocate a new frame (`kalloc()`) and copy the contents of the old frame.
- We map the frame onto the page table of the faulting process with the write flag set, and the instruction restarts.

3. User programs

3.1. **mydemandPage.** : This program's code was copied directly from the assignment document with minimal changes.

It demonstrates that demand paging indeed occurs.

3.2. **myCOW.** : This program demonstrates copy-on-write by printing page tables

- Before fork.
- After fork, in child process, before any writes.
- After writes in child process.

Note: Due to the illegibilty of outputs, the line printing the details of the system calls (from Assignment 5) has been commented out. As a result, the ouput is significantly cleaner.

4. usertests

The correctness of our changes was verified by running the `usertests` binary, which performs exhaustive testing of all functionalities. All tests passed.